

JavaScript

Hintergrund & ECMAScript

Informatik – 3. Semester

1 Hintergrund & ECMAScript

2 Grundlagen, Einbettung & Parsing-Order

Woher kommt JavaScript, und warum braucht es einen Standard?

- Wurde für eingebettete Skripte entwickelt, die einmalig von einem Server heruntergeladen und lokal (im Browser) ausgeführt werden
 - Dadurch lassen sich HTML-Seiten dynamisch verändern
 - Verringert Datenverkehr zwischen Browser und Server
- Typische Aufgaben von JavaScript-Programmen in HTML-Seiten:
 - Eingebene Daten vor dem Versenden prüfen oder formatieren
 - Maus- und Tastatureingaben verarbeiten
 - Dynamische Inhalte wie Suchergebnisse verwalten
- Zunehmend wird es auch für andere Aufgaben verwendet
 - z.B. auch serverseitige Programmierung mit NodeJS

- 1995 wurde die erste Version von JavaScript von der Firma Netscape für ihren damaligen Browser entwickelt
- 1996 erweiterte Microsoft den Internet Explorer um eine weitgehend gleiche Skriptsprache namens JScript
- Ende 1996 begann die Entwicklung eines gemeinsamen Standards für den Kern von JavaScript und JScript: ECMAScript

- „European Computer Manufacturers Association“ ist ein Standardisierungsgremium für Computer und Web-Standards
- Heute heißt es „Ecma International – European association for standardizing information and communication systems.“
- Standards beginnen jedoch heute immer noch mit dem Kürzel ECMA
- Annual Release Cycle (aktuell ES2025)

- ECMAScript ist ein Standard für eine Programmiersprache
- ES definiert Aspekte wie Datentypen, Operatoren, Ausdrücke, Objekte und Kontrollstrukturen
- ES beschreibt jedoch keinerlei Funktionen für die Interaktion mit der ausführenden Plattform
 - z.B. Betriebssystem oder Webbrowser
 - Insbesondere gibt es in ECMAScript keine Ein- und Ausgabefunktionen
- ECMAScript-Code wird deshalb in ein so genanntes „Host Environment“ eingebettet
 - Das Host Environment liefert umgebungsspezifische Features wie Fenster, Menüs, Input & Output, Caching, etc.
 - JavaScript und JScript stellen ein solches Host Environment zur Verfügung
 - Aus Sicherheitsgründen wird der Zugriff auf das Betriebssystem typischerweise stark eingeschränkt



- Wenn JS die einzige aktuell relevante Implementation ist, wozu brauchen wir dann einen getrennten Standard?

- Welche Vorteile und Nachteile entstehen durch eine Trennung von Standard und Implementation?
- Standard = EcmaScript
- Implementation = JavaScript

3 Min

Vorteile

- Paradigmaübergreifende Anwendung
 - Client-Side Logik mit Browsern
 - Server-Side Logik mit Node.js
 - Cross-Plattform Apps mit React Native

Nachteile

- Implementationen variieren
 - Browser haben unterschiedliche Implementationen von JavaScript
 - Dadurch stellen Browser manche Seiten unterschiedlich dar

- V8 (Google)
 - Chrome
 - Chromium
 - Node.js
- SpiderMonkey (Mozilla)
 - Firefox
 - Servo
- Nitro (Apple)
 - Safari
 - WebKit-based browsers
- Chakra (Microsoft)
 - Microsoft Edge (Legacy Version)
 - Windows Apps (HTMLEngine)
 - Deprecated
- Hermes (Facebook)
 - React Native
 - various Facebook apps

- Wir erinnern uns:
 - CSS und HTML sind auch Standards
 - Die Implementierung der Parser ist auch hier von den Browsern abhängig
- Wo kann ich die Verfügbarkeit von Features überprüfen?
 - <https://developer.mozilla.org>
 - <https://caniuse.com/>

- ECMAScript ist ein Standard
 - ES definiert Aspekte wie Datentypen, Operatoren, Ausdrücke, Objekte und Kontrollstrukturen
 - Keine Funktionen zur Interaktion mit Plattformen
 - z.B. keine Ein- und Ausgabefunktionen
- JavaScript ist eine Host Environment für ECMAScript
 - Host Environments liefern umgebungsspezifische Features wie Fenster, Menüs, Input & Output, Caching, etc.

1 Hintergrund & ECMAScript

2 Grundlagen, Einbettung & Parsing-Order

Grundlagen, Einbettung & Parsing-Order

```
<script>
  var targetVariable = "Moin!"
  console.log("Hello there!")
  console.log(typoVariable)
</script>
```

Hello there!

❗ ▶ Uncaught ReferenceError: typoVariable is not defined
 <anonymous> <http://127.0.0.1:5500/v3/examples/variables.htm:4>
 [\[Learn More\]](#)

- JavaScript wird interpretiert, nicht kompiliert
- Lines werden direkt sequenziell ausgeführt, ohne vorher den Gesamtkontext zu ermitteln

```
<script>
  var num = 39;
  var word = "This is a String!"
  word = 2;
  num = "4";
  document.write(word + num);
</script>
```

Ergebnis: 24

- JavaScript verfügt über Variablen und Datentypen
- Im Gegensatz zu Java, sind diese jedoch nicht explizit typisiert
- Der Interpreter erkennt Datentypen bei der Zuweisung aus dem Kontext
 - Das nennt man dynamische Typisierung

```
<script>  
  var x = 3;  
  y = 5;  
  console.log(x + y);  
</script>
```

Ergebnis: 8

- Variablen können deklariert werden, müssen aber nicht
- Es gibt jedoch einen Unterschied

```
<script>
  var x;
  y = x*x;      // y hat den Wert NaN
  y = 2/0;      // y hat den Wert Infinity
  y = -2/0;     // y hat den Wert -Infinity
  y = 0/0;      // y hat den Wert NaN
  y = 1e999;    // y hat den Wert Infinity
  y = 2/(1/0);  // y hat den Wert 0 (!)
</script>
```

- NaN (Not-a-Number)
 - repräsentiert invalide Nummern
- Infinity
 - Repräsentiert Nummern, die größer sind als JavaScripts numerisches Limit

```
<script>  
  var absichtlichLeer = null;  
  var nurLeer;  
</script>
```

- Null
 - Absichtliche Abwesenheit eines Wertes
 - Muss explizit zugewiesen werden
 - Null ist ein Object
- Undefined
 - Automatisch zugewiesener Wert für nicht initialisierte Variablen

Mit var (oder ohne Schlüsselwort) deklarieren Sie eine Variable. Seit ECMA Script 6 können Sie mit let Variablen mit Blockgültigkeit oder mit const Konstanten (final in Java) mit Blockgültigkeit erzeugen.

```
if( true ) {  
    let x = 5;  
}  
console.log(x); //Reference error
```

Lässt sich der folgende Code fehlerfrei ausführen?

```
if( true ) {  
  const x = 5;  
}  
console.log(x);
```

```
<script type="application/javascript">
  function quadrat() {
    var z = document.berechnung.zahl.value;
    var q = z * z;
    alert("Das Quadrat von " + z + " ist " + q);
  }
</script>
<form name="berechnung">
  <input type="text" name="zahl" size="5">
  <input
    type="button"
    value="Quadrat berechnen"
    onclick="quadrat()"
  >
</form>
```

- Mit JavaScript lässt sich auf HTML-Objekte in der DOM zugreifen
- Hier wird auf das Form-Element „berechnung“ zugegriffen

```
<body>
  <script>
    x = 5;
    var tautologie = "X ist immer "
    document.write(tautologie + x);
  </script>
</body>
```

Ergebnis: X ist immer 5

- Das Script-Element erlaubt es, JavaScript code direkt in die HTML-Seite hineinzuschreiben

```
<head>  
  <script src="tautologie.js"></script>  
</head>  
<body>  
</body>
```

Ergebnis: X ist immer 5

- Alternativ lassen sich Skripte auch durch externe Dateien einbinden

- Entwickeln Sie eine Funktion `countSpecificWord(word)` die das Vorkommen eines angegebenen Strings innerhalb eines Containers zählt und das Ergebnis in die entsprechenden `<span>` Elemente schreibt
- Denken Sie daran, vorher die AI-Features Ihrer IDE auszuschalten!
- Lernraumkurs -> Zusatzmaterial -> Übung: countOccurence

15 Min



```
<head>
  <script>console.log("Inline Script!");</script>
  <script src="classic-external.js"></script>
  <script async src="async-script.js"></script>
  <script defer src="defer-script.js"></script>
  <script type="module" src="example-module.mjs"></script>
</head>
<body>
  <script>
    console.log("Dynamically Inserted External Script!");
    const dyn = document.createElement('script');
    dyn.src = 'dynamic-inserted.js';
    document.head.appendChild(dyn);
    const inlineDyn = document.createElement('script');
    inlineDyn.textContent =
      "console.log('Dynamically Inserted Inline Script!');";
    document.body.append(inlineDyn, dyn);
  </script>
</body>
```

- In welcher Reihenfolge werden diese Skripte ausgeführt?



- Skripte können die DOM manipulieren, dementsprechend werden Sie kategorisiert und in strikter Ordnung ausgeführt
- Inline Skripte
- Externe Skripte
- Asynchron-geladene Skripte -> async
- Verzögerte Skripte -> defer
- Modul-Skripte -> type:module
- Dynamisch Insetierte Skripte: